Subexponential Parameterized Odd Cycle Transversal on Planar Graphs

Daniel Lokshtanov[1], Saket Saurabh[2], and Magnus Wahlström[3]

1 **Department of Computer Science and Engineering, University of California, San Diego, USA.** `daniello@ii.uib.no`
2 **The Institute of Mathematical Sciences, Chennai, India.** `saket@imsc.res.in`
3 **Max-Planck-Institut für Informatik, Saarbrücken, Germany.** `wahl@mpi-inf.mpg.de`

Abstract

In the Odd Cycle Transversal (OCT) problem we are given a graph G on n vertices and an integer k, the objective is to determine whether there exists a vertex set O in G of size at most k such that $G \setminus O$ is bipartite. Reed, Smith and Vetta [Oper. Res. Lett., 2004] gave an algorithm for OCT with running time $3^k n^{O(1)}$. Assuming the exponential time hypothesis of Impagliazzo, Paturi and Zane, the running time can not be improved to $2^{o(k)} n^{O(1)}$. We show that OCT admits a randomized algorithm running in $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)} n)$ time when the input graph is planar. As a byproduct we also obtain a linear time algorithm for OCT on planar graphs with running time $O(2^{O(k \log k)} n)$ time. This improves over an algorithm of Fiorini et al. [Disc. Appl. Math., 2008].

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph Theory, Parameterized Algorithms, Odd Cycle Transversal

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.424

1 Introduction

We consider the Odd Cycle Transversal (OCT) problem where we are given as input a graph G with n vertices and m edges, together with an integer k. The objective is to determine whether there exists a vertex set O of size at most k such that $G \setminus O$ is bipartite. This classical optimization problem was proven NP-complete already in 1978 by Yannakakis [28] and has been studied extensively both within approximation algorithms [1, 16] and parameterized algorithms [12, 17, 20, 22, 24, 26].

It was a long-standing open problem whether OCT is *fixed-parameter tractable (FPT)*, that is solvable in time $f(k)n^{O(1)}$ for some function f depending only on k. In 2004 Reed, Smith and Vetta [26] resolved the question positively, and gave an $O(4^k kmn)$ time algorithm for the problem. It was later observed by Hüffner [17] that the running time of the algorithm of Reed et al. is actually $O(3^k kmn)$.

Improving over the algorithm of Reed et al. [26], both in terms of the dependence on k, and in terms of the dependence on input size remain interesting research directions. For the dependence on input size, Reed et al. [26] point out that using techniques from the Graph Minors project of Robertson and Seymour one could improve the nm factor in the running time of their algorithm to n^2, at the cost of worsening the dependence on k. They pose the existence of a linear time algorithm for OCT for every fixed value of k as an open problem. Fiorini et al [12] showed that if the input graph is required to be planar, then OCT has a $2^{O(k^6)}n$ time algorithm. Later Kawarabayashi and Reed [20] gave an "almost" linear time

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 424–434
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algorithm for OCT, that is an algorithm with running time $f(k)n\alpha(n)$ where $\alpha(n)$ is the inverse ackermann function and f is some computable function of k.

When it comes to the dependence of the running time on k, the $O(3^k nm)$ algorithm of Reed et al. [26] remained the best known until a recent manuscript of Lokshtanov et al. [24] (see also Narayanaswamy et al. [25]) giving an algorithm with running time $O(2.32^k n^{O(1)})$ using linear programming techniques. It is tempting to ask how far down one may push the dependence of the running time on k. Should we settle for c^k for a reasonably small constant c, or does there exist a *subexponential* parameterized algorithm for OCT, that is an algorithm with running time $2^{o(k)}n^{O(1)}$? It turns out that assuming the *Exponential Time Hypothesis* of Impagliazzo, Paturi and Zane [18] there can not be a subexponential parameterized algorithm for OCT. In this paper we show that restricting the input to planar graphs circumvents this obstacle – in particular we give an $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)}n)$ time algorithm for OCT on planar graphs (we will refer to OCT on planar graphs as Pl-OCT). As a corollary of our main result we also obtain a simple $O(k^{O(k)}n)$ time algorithm for Pl-OCT, improving over the dependence on k in the algorithm of Fiorini et al. [12] while keeping the linear dependence on n.

Methods. There are many NP-complete graph problems that remain NP-complete even when restricted to planar graphs [15] but admit much better approximation algorithms and faster parameterized algorithms on planar graphs than on general graphs. The *bidimensionality theory* of Demaine et al. [7, 10] aims to explain this phenomenon. Specifically, using bidimensionality one can give fast parameterized algorithms [6], approximation schemes [8, 13] and efficient polynomial time pre-processing algorithms [14], called kernelization algorithms, for a host of problems on planar graphs, and more generally on classes excluding a forbidden minor. The main driving force behind bidimensionality is that for many parameterized problems on planar graphs one can bound the *treewidth* of the input graph as a *sublinear* function of the parameter k. For some problems, including OCT, this approach seems not to be amenable as there is no apparent connection between the parameter k and the treewidth of the input graph. Nevertheless, a variant of this idea is still the engine of the subexponential time parameterized algorithms of Dorn et al. [9] and Tazari [27], the linear time algorithm of Fiorini et al. [12] and also of our algorithm.

Fiorini et al. show that after a linear time pre-processing step, the treewidth of the input graph is bounded by $O(k^2)$. Well-known algorithms for finding tree decompositions [3] and an algorithm for OCT on graphs of bounded treewidth then do the job. To obtain our $O(k^{O(k)}n)$ time algorithm for Pl-OCT, we give a linear time *branching* step inspired by Baker's layering approach [2] that produces $O(k)$ instances, each of treewidth $O(k)$, such that the input instance is a "yes" instance if and only if at least one of the output instances is a "yes" instance. We then show that one can make a trade-off between the number of output instances of the branching process and the treewidth of the output graphs. In particular we show that we can output $k^{O(\sqrt{k})}$ instances each of treewidth $O(\sqrt{k})$, such that the input instance is a "yes" instance if and only if at least one of the output instances is a "yes" instance. The parameters involved in this trade-off are rather delicate, and to make the trade-off go through we need to first pre-process the graph using the recent sophisticated methods of Kratsch and Wahlström [22, 23]. This pre-processing step is the only part of our algorithm which takes superlinear time, and so we obtain an algorithm with running time $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)}n)$. It remains an interesting open problem whether there is a subexponential parameterized algorithm for Pl-OCT with linear dependence on n.

2 Preliminaries

Throughout this paper we use n to denote the size of the vertex set of the input graph G. For a graph G we denote its vertex set by $V(G)$ and the edge set by $E(G)$. An edge between vertices u and v is denoted by uv, and is identical to the edge vu. We use $G[V']$ to denote the subgraph of G induced by V', i.e., the graph on vertex set V' and edge set $\{uv \in E(G) \mid u, v \in V'\}$. We use $G \setminus Z$ as an abbreviation for $G[V(G) \setminus Z]$. The open neighborhood of a vertex v in graph G contains the vertices adjacent to v, and is written as $N_G(v)$. The open neighborhood of a set $S \subseteq V(G)$ is defined as $\bigcup_{v \in S} N_G(v) \setminus S$. We omit the subscript G when it is clear from the context. A graph G is *bipartite* if there exists a partition of $V(G)$ into two sets A and B such that every edge of G has one endpoint in A and one in B. The sets A and B are called bipartitions of G. A set W of $V(G)$ is called an *odd cycle transversal* of G if $G \setminus W$ is bipartite. A *plane* embedding of a graph G is an embedding of G in the plane with no edge crossings. A graph G that has a plane embedding is called *planar*. A *plane* graph is a graph G together with a plane embedding of it. For a plane graph G, $F(G)$ is the set of faces of G.

2.1 Tree-width

Let G be a graph. A *tree decomposition* of a graph G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$ (here T is a tree) such that

1. $\bigcup_{t \in V(T)} X_t = V(G)$,
2. for every edge $\{x, y\} \in E(G)$ there is a $t \in V(T)$ such that $\{x, y\} \subseteq X_t$, and
3. for every vertex $v \in V(G)$ the subgraph of T induced by the set $\{t \mid v \in X_t\}$ is connected.

The *width* of a tree decomposition is $\left(\max_{t \in V(T)} |X_t|\right) - 1$ and the *treewidth* of G is the minimum width over all tree decompositions of G. We use $\mathbf{tw}(G)$ to denote the treewidth of the input graph G.

A tree decomposition $(T, \mathcal{X})$ is called a *nice tree decomposition* if T is a tree rooted at some node r where $X_r = \emptyset$, each node of T has at most two children, and each node is of one of the following kinds:

1. *Introduce node*: a node t that has only one child t' where $X_t \supset X_{t'}$ and $|X_t| = |X_{t'}| + 1$.
2. *Forget node*: a node t that has only one child t' where $X_t \subset X_{t'}$ and $|X_t| = |X_{t'}| - 1$.
3. *Join node*: a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.
4. *Leaf node*: a node t that is a leaf of t, is different than the root, and $X_t = \emptyset$.

Notice that, according to the above definition, the root r of T is either a forget node or a join node. It is well-known that any tree decomposition of G can be transformed into a nice tree decomposition in time $O(|V(G)| + |E(G)|)$ maintaining the same width [21]. We use G_t to denote the graph induced on the vertices $\bigcup_{t'} X'_t$, where t' ranges over all descendants of t, including t. We use H_t to denote $G_t[V(G_t) \setminus X_t]$.

3 Subexponential Time FPT Algorithm for Pl-OCT

In this section we outline our algorithms for Pl-OCT – (a) an algorithm running in time $O(k^{O(k)}n)$ and (b) an algorithm running in time $O(n^{O(1)} + 2^{O(\sqrt{k}\log k)}n)$. To do so we reduce the problem to a "Steiner tree-like" problem on graphs of small treewidth and then use an algorithm for this Steiner tree-like problem on graphs of bounded treewidth to obtain our results.

3.1 Reducting Pl-OCT to a "Steiner tree-like" problem

It is well-known that a plane graph is bipartite if and only if every face is even. Here we say that a face is even if the cyclic walk enclosing the face has even length. This fact allows us to interpret the OCT problem on a plane graph G as the "Steiner tree-like" L-Join problem on the face-vertex incidence graph of G. The *face-vertex incidence graph* of a plane graph G is the graph G^+ with vertex set $V(H) = V(G) \cup F(G)$ and an edge between a face $f \in F(G)$ and vertex $v \in V(G)$ if v is incident to f in the embedding of G. Clearly G^+ is planar, and also it is bipartite with bipartitions $V(G)$ and $F(G)$. For subsets $L \subseteq F(G)$ and $O \subseteq V(G)$ we will say that O is an L-*join* in G^+ if every connected component of $G^+[F(G) \cup O]$ contains an even number of vertices from L. The following observation plays a crucial role in our algorithm.

▶ **Proposition 1** ([12]). A subset O of $V(G)$ is an odd cycle transversal of G if and only if every connected component of $G^+[F(G) \cup O]$ has an even number of vertices of L. Here, L is the set of odd faces of G.

Observe that the notion of an L-join can be defined for any bipartite graph H with bipartitions A and B. Specifically for subsets $L \subseteq A$ and $O \subseteq B$ we say that O is an L-join in H if every connected component of $H[A \cup O]$ contains an even number of vertices from L. In the L-Join problem we are given a bipartite graph H with bipartitions A and B, together with a subset $L \subseteq A$ and an integer k. The task is to determine whether there is an L-join $W \subseteq B$ in H of size at most k. The Pl-L-Join problem is just L-Join, but with the input graph H required to be planar. Proposition 1 directly implies the following lemma.

▶ **Lemma 2.** *If there is an algorithm for* Pl-L-Join *with running time $O(f(k)n^c)$ for a function f and constant $c \geq 1$ then there is an algorithm for* Pl-OCT *with running time $O(f(k)n^c)$.*

In Section 3.2 we will give an algorithm for Pl-L-Join with running time $O(2^{O(k \log k)} n)$, yielding an algorithm for Pl-OCT with the same running time. To get a subexponential time algorithm for Pl-OCT we will reduce to a *promise* variant of Pl-L-Join where we additionally are given a set S of size $k^{O(1)}$ with the promise that an optimal solution can be found inside S. We now formally define the promise variant of Pl-L-Join that we will reduce to.

Promise Planar-L-Join (PrPl-L-Join)

Input:	A bipartite planar graph H with bipartitions A and B, a set of terminals $L \subseteq A$, a set of annotated vertices $S \subseteq B$ and an integer k
Parameter:	$\lvert S\rvert$, k
Question:	Is there an L-join $O \subseteq B$ of size at most k?
Promise:	If an L-join $O \subseteq B$ of size at most k exists then there is an L-join $O' \subseteq S$ of size at most $\lvert O\rvert$.

In order to be able to reduce Pl-OCT to PrPl-L-Join we show the following lemma.

▶ **Lemma 3** (Small Relevant Set Lemma). *Let (G, k) be a yes instance to* Pl-OCT. *Then in polynomial time we can find a set S such that*
- $|S| = k^{O(1)}$; *and*
- *with probability $(1 - \frac{1}{2^n})$, G has an oct of size k if and only if there is an oct contained in S of size k.*

Here $n = |V(G)|$.

Proof. This follows from [22, 23], but for completeness we sketch the proof here. First, we find in polynomial time an approximate solution of size at most $\frac{9}{4}k$ by applying the $\frac{9}{4}$-approximation algorithm for PL-OCT by Goemans and Williamson [16]. Let X be such an approximate solution. Next, we create an auxiliary graph G' from G and X as in the algorithm of Reed, Smith, and Vetta [26]; the vertex set of G' is $(V \setminus X) \cup X'$, where X' is a set of $2|X|$ terminals corresponding to X. It is a consequence of [26], made explicit in [22, Lemma 4.1], that a minimum oct can be found by taking the union of a subset of X and a minimum S-T vertex cut in $G' \setminus R$ for $S, T, R \subseteq X'$ (it may be assumed that all minimum cuts are disjoint from X', by modifying R). By [23, Corollary 1], there exists a set $Z \subseteq V(G')$ with $|Z| = O(|X|^3)$ which includes such a min-cut for all choices of S, T, R, and we can find it in polynomial time, with success probability as stated, using the tools of representative sets from matroid theory; see [23]. ◀

Proposition 1 together with Lemma 3 directly imply the following lemma.

▶ **Lemma 4.** *If there is an algorithm for* PRPL-L-JOIN *with running time $O(f(k)n^c)$ for a function f and constant $c \geq 1$ then there is a randomized algorithm for* PL-OCT *with running time $O(n^{O(1)} + f(k)n^c)$ and success probability at least $(1 - \frac{1}{2^n})$.*

At this point we make a remark about results in [22, 23]. In [22, 23], Kratsch and Wahlström obtain a polynomial kernel for OCT. That is, given an input (G, k) they output an equivalent instance (G', k') such that G has an odd cycle transversal of size k if and only if G' has and $k' \leq k$. It is very tempting to use this result directly at the place of Lemma 3. However, for our subexponential algorithm for PL-OCT we not only need that $k' \leq k$, G' has small size but also that G' is a planar graph. However, it is not clear that the algorithms described in [22, 23] could be easily modified to get both $k' \leq k$ and G' is planar. Thus we resort to Lemma 3 which is sufficient for our purpose.

3.2 Algorithms for PL-L-JOIN, PRPL-L-JOIN and PL-OCT

In this section we will give fast parameterized algorithms for PL-L-JOIN and PRPL-L-JOIN. The algorithms are based on the following decomposition lemma.

▶ **Lemma 5.** *There is an algorithm that given a planar bipartite graph H with bipartitions A and B and an integer t, runs in time $O(n)$ and computes a partition of B into $B = B_1 \cup B_2 \ldots \cup B_t$ such that $\mathbf{tw}(G \setminus B_i) = O(t)$ for every $i \leq t$. Furthermore, for every $i \leq t$ a tree-decomposition of $G \setminus B_i$ of width $O(t)$ can be computed in time $O(tn)$.*

Proof. Select a vertex $r \in A$ and do a breadth first search in H starting from r. We call $\{r\}$ the first BFS layer, $N(r)$ the second BFS layer, $N(N(r)) \setminus \{r\}$ the third BFS layer etc. Let $L_1, L_2, \ldots, L_\ell$ be the BFS layers of H. Since H is bipartite we have that for every odd i, $L_i \subseteq A$ while for every even i we have $L_i \subseteq B$. For every i from 1 to t set $B_i = \bigcup_{j \geq 0} L_{2i+2tj}$. It is easy to see that $B_1, \ldots, B_t$ indeed form a partition of B. Furthermore, for every i, every connected component C of $H \setminus B_i$ is a subset of at most $2t$ consecutive BFS layers of H. Contracting all of the BFS layers preceeding C in H into a single vertex shows that C is an induced subgraph of a planar graph of diameter $O(t)$. Thus it follows from [4, 11] that a tree decomposition of C of width $O(t)$ can be computed in time $O(t|C|)$. Hence for every $i \leq t$ a tree-decomposition of $G \setminus B_i$ of width $O(t)$ can be computed in time $O(tn)$. ◀

In Section 4 we will prove the following lemma.

▶ **Lemma 6.** *There is an algorithm that given an bipartite graph H with bipartitions A and B, together with a set $L \subseteq A$, an integer k and a tree-decomposition of H of width w, determines whether there is an L-join $W \subseteq B$ of size at most k in time $O(w^{O(w)}n)$.*

Lemmata 5 and 6 yield the $O(2^{O(k \log k)}n)$ time algorithm for Pl-L-Join.

▶ **Lemma 7.** *There is a $O(2^{O(k \log k)}n)$ time algorithm for* Pl-L-Join.

Proof. Given as input a planar bipartite graph H with bipartitions A and B, a set $L \subseteq A$ and an integer k the algorithm applies Lemma 5 with $t = k + 1$. Now, if H has an L-join W of size at most k then there is an $i \leq t$ such that $W \cap B_i = \emptyset$, and so W is an L-join in $H \setminus B_i$. Furthermore, for any j an L-join in $H \setminus B_j$ is also an L-join in H. We loop over every i and return the smallest L-join of $H \setminus B_i$. By Lemma 5, for each i we can compute a tree-decomposition of $H \setminus B_i$ of width $O(t)$ in $O(tn)$ time. By Lemma 6 we can find a smallest L-join of $H \setminus B_i$ in time $O(2^{O(k \log k)}n)$. ◀

The algorithm for PrPl-L-Join goes along the same lines as the algorithm in Lemma 7, but is slightly more involved.

▶ **Lemma 8.** *There is an $O(|S|^{\sqrt{k}} \cdot 2^{O(\sqrt{k} \log k)} \cdot n)$ time algorithm for* PrPl-L-Join.

Proof. Given as input a planar bipartite graph H with bipartitions A and B, a set $L \subseteq A$ of terminals and a set $S \subseteq B$ of annotated vertices together with an integer k the algorithm applies Lemma 5 with $t = \sqrt{k}$. For every $i \leq t$ define $W_i = W \cap B_i$. Now, if H has an L-join W of size at most k then without loss of generality $W \subseteq S$. Furthermore there is an $i \leq t$ such that $|W_i| \leq \sqrt{k}$. Observe that W is also an L-join in $H \setminus (B_i \setminus W_i)$. Furthermore, for any subset B' of B, an L-join in $H \setminus B'$ is also an L-join in H. The algorithm loops over every i, and every choice of $W_i^* \subseteq B_i \cap S$ with $|W_i^*| \leq \sqrt{k}$. There are $\sqrt{k}$ choices for i and at most $|S|^{\sqrt{k}}$ choices for W_i^*. For each choice of i and W_i^* the algorithm finds the smallest L-join of $H \setminus (B_i \setminus W_i^*)$. Correctness follows from the fact that we will loop over the choice $W_i^* = W_i$.

In order to find the smallest L-join of $H \setminus (B_i \setminus W_i^*)$ we will apply Lemma 6, but in order to do that we need a tree decomposition of $H \setminus (B_i \setminus W_i^*)$ of small width. However, by Lemma 5 we can find a tree decomposition of $H \setminus B_i$ of width $O(\sqrt{k})$ in linear time for every i. Adding W_i^* to every bag of this tree decomposition yields a tree decomposition of $H \setminus (B_i \setminus W_i^*)$ of width $O(\sqrt{k}) + |W_i^*| = O(\sqrt{k})$. Thus, by Lemma 6 we can find the smallest L-join of $H \setminus (B_i \setminus W_i^*)$ in time $O(2^{O(\sqrt{k} \log k)} \cdot n)$ for every choice of i and W_i^*. Since there are $|S|^{\sqrt{k}}$ choices for W_i and $\sqrt{k}$ choices for i this concludes the proof. ◀

We are now ready to prove our main theorems. In particular, Lemmata 2 and 7 imply our linear time parameterized algorithm for Pl-OCT.

▶ **Theorem 9.** *There is a $O(2^{O(k \log k)}n)$ time algorithm for* Pl-OCT.

Similarly, Lemmata 4 and 8 imply our subexponential parameterized algorithm for Pl-OCT.

▶ **Theorem 10.** *There is an $O(n^{O(1)} + 2^{O(\sqrt{k} \log k)}n)$ time randomized algorithm for* Pl-OCT.

4 An algorithm for Minimum L-Join on graphs of bounded treewidth

In this section we give a dynamic programming algorithm on graphs of bounded treewidth for the following problem.

Minimum L-Join	
Input:	A bipartite graph G with bipartitions C and D and a set $L \subseteq C$.
Parameter:	$\mathbf{tw}(G)$
Question:	Find a minimum sized set $W \subseteq D$ (if it exists) such that every connected component of $G[C \cup W]$ has an even number of vertices of L.

Observe that finding W is equivalent to finding a forest F of G such that $L \subseteq V(F)$ and each tree of F contains an even number of vertices of L.

4.1 Description of the Algorithm

The idea of our algorithm is to do dynamic programming starting from leaf to root. We set

$$|X_t| = w \qquad L_t = L \cap V(G_t) \qquad C_t = C \cap V(G_t)$$

For a node t and any solution, say F, intersection with G_t and X_t (a partial solution) could be described as follows:

- A tree F_i of F is contained inside $V(G_t) \setminus X_t$ and in this case we have that $|V(F_i) \cap L|$ is even.
- A tree F_i of F does not contain any vertex of $V(G_t)$.
- A tree F_i of F contains vertices from both $V(G_t)$ and $V(G) \setminus V(G_t)$. In this case we have that F_i contains vertices from X_t and either contains an even or an odd number of vertices from L.

We would like to keep representatives for all partial solutions for the graph G_t. Towards this we first introduce the following definition.

▶ **Definition 11.** A set P is a *partition* of X if, and only if, it does not contain the empty set unless $X = \emptyset$ and: (a) the union of the elements of P is equal to X; and (b) the intersection of any two elements of P is empty. (We say the elements of P are pairwise disjoint.) We call an element of P as *piece*. A partition is called *signed partition* if for every piece $A \in P$, we assign either 0 or 1. The sign of a piece A is denoted by $sign(A)$. That is, $sign$ is a function from P to $\{0, 1\}$. A signed partition is denoted by $(P, sign)$, that is, a pair consisting of the partition P and a function $sign : P \to \{0, 1\}$.

For each node $i \in V(T)$ we compute a table A_i, the rows of which are 3-tuples $[S, (P, sign), val]$. Table A_i contains one row for each combination of the first two components which denote the following:

- S is a subset of X_i.
- $(P, sign)$, where P is a partition of S into at most $|S|$ labelled pieces.

We use $P(v)$ to denote the piece of the partition P that contains the vertex v. We let $|P|$ denote the number of pieces in the partition P. The set S denotes the intersection of our solution with the vertices in the bag X_i.

The last component val, also denoted as $A_i[S, (P, sign)]$, is the size of a smallest forest $F_i(S, (P, sign))$ of G_i which satisfies the following properties:

- $C_i \subseteq V(F_i(S, (P, sign)))$ – all the vertices of C lying in G_i are contained in the forest;
- $(X_i \setminus S) \cap V(F_i(S, (P, sign))) = \emptyset$ – only vertices in S from X_i are contained in the forest;
- for every non-empty part A of P there exists a tree, say F_A in $F_i(S, (P, sign))$, such that $A \subseteq V(F_A)$ and $|L_i \cap V(F_A)| \bmod 2 = sign(A)$ and for every $A \neq B$, $F_A \neq F_B$ (that is, trees associated with distinct parts are distinct); and

- if there exists a tree F'' in $F_i(S, (P, sign))$ such that $V(F'') \cap X_i = \emptyset$ then $|L_i \cap V(F'')| \bmod 2 = 0$.

If there is no such forest $F_i(S, (P, sign))$, then the last component of the row is set to ∞. Given a node i of the tree T and a pair $(S, (P, sign))$ of X_i, a forest F in G_i satisfying the above properties is called *consistent* with $(S, (P, sign))$.

We compute the tables A_i starting from the leaf nodes of the tree decomposition and going up to the root.

Leaf Nodes. Let i be a leaf node of the tree decomposition. We compute the table A_i as follows. We set $A_i[\emptyset, (\emptyset, 0)] = 0$ and $A_i[\emptyset, (\emptyset, 1)] = 0$.

Introduce Nodes. Let i be an introduce node and j its unique child. Let $x \in X_i \setminus X_j$ be the introduced vertex. For each pair $(S, (P, sign))$, we compute the entry $A_i[S, (P, sign)]$ as follows.

Case 1. $x \in S$. Check whether $N(x) \cap S \subseteq P(x)$; if not, set $A_i[S, (P, sign)] = \infty$.

Subcase 1: $P(x) = \{x\}$. If ($x \in L_i$ and $sign(P(x)) = 0$) or ($x \notin L_i$ and $sign(P(x)) = 1$) then set $A_i[S, (P, sign)] = \infty$.
Else, we set $A_i[S, (P, sign)] = A_j[S \setminus \{x\}, (P \setminus P(x), sign')] + 1$. Here $sign'$ is the restriction of $sign$ to $P \setminus P(x)$.

Subcase 2: $|P(x)| \geq 2$ and $N(x) \cap P(x) = \emptyset$. Set $A_i[S, (P, sign)] = \infty$, as no extension of $P(x)$ in G_i is connected.

Subcase 3: $|P(x)| \geq 2$ and $N(x) \cap P(x) \neq \emptyset$. Let $\mathcal{A}$ be the set of all rows $[S', (P', sign')]$ of the table A_j that satisfy the following conditions:

- $S' = S \setminus \{x\}$.
- $P' = (P \setminus P(x)) \cup Q$, where Q is a partition of $P(x) \setminus \{x\}$ such that each piece of Q contains an element of $N(x) \cap P(x)$.
- $sign'$ is such that it agrees with $sign$ on $P \setminus P(x)$ and if $x \in L_i$ then

$$\left(1 + \sum_{Q_\ell \in Q} sign'(Q_\ell)\right) \bmod 2 = sign(P(x),$$

else

$$\left(\sum_{Q_\ell \in Q} sign'(Q_\ell)\right) \bmod 2 = sign(P(x)).$$

Set $A_i[S, (P, sign)] = \min_{[S', (P', sign')] \in \mathcal{A}} \left\{ A_j[S', (P', sign')] \right\} + 1$.

Case 2. $x \notin S$. If $x \in C_i$ then set $A_i[S, (P, sign)] = \infty$. Else set $A_i[S, (P, sign)] = A_j[S, (P, sign)]$.

Forget Nodes. Let i be a forget node and j its unique child node. Let $x \in X_j \setminus X_i$ be the forgotten vertex. For each pair $(S, (P, sign))$ in the table A_i, let $\mathcal{A}$ be the set of all rows $[S', (P', sign')]$ of the table A_j that satisfy the following conditions:

- $S' = S \cup \{x\}$, and
- $P'(x) = P(y) \cup \{x\}$ for some $y \in S$ and all other parts remain the same. Essentially, P' has been obtained by adding x to some part of P.
- $sign'$ is same as $sign$ on all other parts of P' but $P'(x)$ and $sign(P'(x)) = sign(P(y))$.

Set

$$A_i[S,(P,sign)] = \min_{[S',(P',sign')]\in\mathcal{A}} \Big\{A_j[S',(P',sign')]\Big\}.$$

Join Nodes. Let i be a join node and j and l its children. For each triple $(S,(P,sign))$ we compute $A_i[S,(P,sign)]$ as follows.
Let $\mathcal{A}$ denote the set of all pairs $\langle(S,(P_1,sign_1)),(S,(P_2,sign_2))\rangle$, where $(S,(P_1,sign_1)) \in A_j$ and $(S,(P_2,sign_2)) \in A_l$ with the following property:

Starting with the partitions $Q_p = P_1$ and the sign function $sign_p = sign_1$ and repeatedly applying the following operation, we reach the stable partition that is identical to $(P,sign)$. The operation that we apply is:

If there exist vertices $u,v \in S$ such that they are in different pieces of Q_p but are in the same piece of P_2, delete $Q_p(u)$ and $Q_p(v)$ from Q_p and add $Q_p(u)\cup Q_p(v)$. Furthermore make $sign_p(Q_p(u)\cup Q_p(v)) := (sign_p(P(u)) + sign_p(P(v))) \bmod 2$.

Set

$$A_i[S,(P,sign)] = \min_{\langle(S,(P_1,sign_1)),(S,(P_2,sign_2))\rangle\in\mathcal{A}} \Big\{A_j[S,(P_1,sign_1)]+A_l[S,(P_2,sign_2)]-|S|\Big\}.$$

The stated conditions ensure that $u,v \in S$ are in the same piece of P if and only if for each $\langle(S,(P_1,sign_1)),(S,(P_2,sign_2))\rangle \in \mathcal{A}$, they are in the same piece of P_1 or of P_2 (or both). Given this, it is easy to verify that the above computation correctly determines $A_i\,[S,(P,sign)]$.

Root Node. We obtain the size of a smallest L-join of G from any row of the table A_r for the root node r. That is, if the size of the forest we have stored is η, then the size of the smallest L-join of G is $\eta - |C|$.

Extracting the solution at the root node. We can compute the optimum solution, that is the set W, by standard backtracking or by storing a set of vertices for each row and each bag.

4.2 Correctness and the Time analysis of the algorithm

We are now ready to discuss the algorithm's running time and prove that it correctly computes an optimal solution.

Proof. (of Lemma 6) We first upper-bound the running time of the algorithm we described earlier. The running time mainly depends on the size of the tables and the combination of tables during the bottom-up traversal of the decomposition tree. Let ζ be the size of the number of signed partitions of size at most $w+1$. The number ζ is upper bounded by $(w+1)^{w+1} \times 2^{w+1}$. Thus the size of the table at any node is upper bounded by $2^{w+1} \times \zeta = 4^{w+1}(w+1)^{w+1} = w^{O(w)}$. Furthermore time taken to compute the value for any row is upper bounded by $w^{O(w)}$. Thus the total time taken by the algorithm is upper bounded by $w^{O(w)} \cdot n = 2^{O(w \log w)} \cdot n$.

The algorithm's correctness can be shown by a standard inductive proof on the decomposition tree. This completes the proof. For an example see [5] for similar proof for the Steiner tree problem parameterized by treewidth of the input graph. ◀

5 Open Problems and Conclusions

In this paper we gave the first subexponential time algorithm for Pl-OCT combining the recent matroid based kernelization for OCT and a reformulation of Pl-OCT in terms of T-joins. On the way we also obtained an algorithm for Pl-OCT running in time $O(k^{O(k)}n)$, improving over the previous linear time FPT algorithm for Pl-OCT by Fiorini et al. [12]. Let us remark that Fiorini et al. [12] do not compute the dependence of the running time on k of their algorithm, and the running time of their algorithm depends on how one implements a particular step, where one needs to compute a tree-decomposition of width $O(k^2)$ of a particular planar graph. Naively using Bodlaender's algorithm [3] gives an $O(2^{O(k^6)}n)$ time algorithm. By using more clever tricks, such as using Kammer and Tholey's [19] recent linear time constant factor approximation algorithm for treewidth of planar graphs, one may get an $O(2^{O(k^2)}n)$ time algorithm. This is still quite a bit slower than our $O(k^{O(k)}n)$ running time.

We conclude with two interesting problems that remain open. First, is there a subexponential parameterized algorithm for Pl-OCT with linear dependence on n? Second, is there an algorithm for Pl-OCT running in time $2^{O(\sqrt{k})}n^{O(1)}$?.

References

1 A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In *STOC*, pages 573–581, 2005.

2 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

3 H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

4 H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

5 M. Chimani, P. Mutzel, and B. Zey. Improved Steiner tree algorithms for bounded treewidth. In *IWOCA*, volume 7056, pages 374–386, 2011.

6 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *J. ACM*, 52(6), 2005.

7 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

8 E. D. Demaine and M. T. Hajiaghayi. Bidimensionality: new connections between FPT algorithms and PTASs. In *SODA*, pages 590–601, 2005.

9 F. Dorn, F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs. In *STACS*, pages 251–262, 2010.

10 F. Dorn, F. V. Fomin, and D. M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.

11 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

12 S. Fiorini, N. Hardy, B. A. Reed, and A. Vetta. Planar graph bipartization in linear time. *Discrete Applied Mathematics*, 156(7):1175–1180, 2008.

13 F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Bidimensionality and EPTAS. In *SODA*, pages 748–759, 2011.

14 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In *SODA*, pages 503–510, 2010.

15 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 M. X. Goemans and D. P. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica*, 18(1):37–59, 1998.

17 F. Hüffner. Algorithm engineering for optimal graph bipartization. *J. Graph Algorithms Appl.*, 13(2):77–98, 2009.

18 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

19 F. Kammer and T. Tholey. Approximate tree decompositions of planar graphs in linear time. In *SODA*, pages 683–698, 2012.

20 K. Kawarabayashi and B. A. Reed. An (almost) linear time algorithm for odd cyles transversal. In *SODA*, pages 365–378, 2010.

21 T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

22 S. Kratsch and M. Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In *SODA*, pages 94–103, 2012.

23 S. Kratsch and M. Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS*, 2012. To appear. Preprint available at `http://arxiv.org/abs/1111.2195`.

24 D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. Faster parameterized algorithms using linear programming. *CoRR*, abs/1203.0833, 2012.

25 N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. LP can be a cure for parameterized problems. In *STACS*, pages 338–349, 2012.

26 B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

27 S. Tazari. Faster approximation schemes and parameterized algorithms on h-minor-free and odd-minor-free graphs. In *MFCS*, pages 641–652, 2010.

28 M. Yannakakis. Node- and edge-deletion NP-complete problems. In *STOC*, pages 253–264, 1978.